Programming Example:

The example program shown below illustrates several points.

1) Binary search algorithm
2) Use of table of pointers pointing to variable length data
3) Use of registers to pass parameters to subroutines
4) Using push and pop to preserve registers modified by a subroutine to preserve
   the register state of the calling procedure

Binary Search:

Binary search is a search algorithm that can be employed to search ordered (sorted) data. It depends upon the fact that the data is ordered to work. The algorithm works by dividing the data set into smaller and smaller regions until the target item is either found, or found not to be in the set. Two pointers are maintained into the data set. One points to the lower end of the range being considered, the other points to the upper end of the range.

At each iteration of the algorithm, the item at the middle of the range is compared to the target value. If the item matches, then the search terminates with success. If the item doesn’t match then the range is adjusted by either moving the lower pointer up or the upper pointer down. The determination of which pointer to move is made by whether the item tested was greater than or less than the item being search for. This traps the item being searched for into smaller and smaller ranges until it is found, or the size of the range goes to 0, in which case the item isn’t in the data set.

Use of pointers for variable length data

When maintaining an array of data items of variable length, it is generally less useful to maintain them directly in the table. There are several reasons for this. If items are being inserted and deleted from the table, it makes the insertion and deletion task more difficult if the items in the table are not all the same size. If reordering the items in the table is required (such as when sorting) the task or exchanging items is much more difficult if all items are not the same size. Generally when maintaining a table of large or variable sized items, an array of pointers to the items is maintained. Then, for example, when sorting the data, it is only necessary to exchange pointers to swap items rather than moving all of the data associated with the item.
; FindStr -- This is an example program to illustrate a
binary search
; of a table of pointers to strings.

_DATA segment public byte 'DATA'

str0 db "This is string 00",0
str1 db "This is string 01",0
str2 db "This is string 02",0
str3 db "This is string 03",0
str4 db "This is string 04",0
str5 db "This is string 05",0
str6 db "This is string 06",0
str7 db "This is string 07",0
str8 db "This is string 08",0
str9 db "This is string 09",0
str10 db "This is string 10",0
str11 db "This is string 11",0
str12 db "This is string 12",0
str13 db "This is string 13",0
str14 db "This is string 14",0

strtab dw str0,str1,str2,str3,str4,str5,str6,str7
     dw str8,str9,str10,str11,str12,str13,str14

srch0 db "Not in table",0
srch1 db "This is string 03",0
srch2 db "This is string 11",0
srch3 db "This is string 00",0
srch4 db "This is string 14",0
srch5 db "Also not in table",0

_DATA ends

_STCK segment stack

dw 128 dup (?)

_STCK ends
_TEXT    segment public byte 'CODE'
    assume   cs:_TEXT

start:
    mov    ax,_DATA
    mov    ds,ax
    mov    es,ax
    assume   ds:_DATA,es:_DATA

    mov    bx,offset strtab
    mov    cx,15

    mov    dx,offset srch0
    call   FindStr

    mov    dx,offset srch1
    call   FindStr

    mov    dx,offset srch2
    call   FindStr

    mov    dx,offset srch3
    call   FindStr

    mov    dx,offset srch4
    call   FindStr

    mov    dx,offset srch5
    call   FindStr

    mov    ax,4C00h
    int    21h
; FindStr - Perform a binary search of a table of a sorted
; table of strings
; for a particular string.
;
; Input:
; BX   - base address of the string pointer table
; CX   - number of string pointers in the table
; DX   - pointer to the search string
; Output:
; AX   - returns the index of the search string
; CF   - clear if string found, set if not found
;
FindStr  proc near
    push cx
    push dx
    push si
    push di

    mov di,dx  ;keep target str ptr in DI
    mov dx,cx  ;keep high pointer in DX
    xor cx,cx  ;keep low pointer in CX

    ; Find the middle of the current range, and compare that
    ; string with the target string
    fnds20: cmp cx,dx  ;if low ptr >= high ptr
           jae fnds80  ; then string isn't in table

           mov ax,cx
           add ax,dx
           shr ax,1     ;find middle of range
           mov si,ax
           add si,si
           mov si,[bx+si] ;get pointer to string at
                           ;middle to SI
           call CmpStr  ;compare the strings
           jz fnds90    ;if match, we found it

           ; This one wasn't the one. Adjust the low end up or the
           ; high end down.
           jc fnds30    ;if *si < *di adjust low end up
           mov dx,ax
           jmp fnds20

    fnds30: mov cx,ax
           jmp fnds20

FindStr  endp

CmpStr   proc near
    push bp
    mov bp,sp
    mov word ptr [bp+6],cx
    mov word ptr [bp+8],dx
    mov word ptr [bp+10],bx
    mov word ptr [bp+12],si
    mov word ptr [bp+14],di
    mov al,[bp+16]
    mov bl,[bp+18]
    cmp al,bl
    je fnds10
    jg fnds20
    pop bp
    ret

fnds10:
    pop bp
    ret

fnds20:
    pop bp
    ret

CmpStr   endp
; String not found in the table. Return CF set
fnds80:  stc
;
; All done
fnds90:  pop di
    pop si
    pop dx
    pop cx
    ret
FindStr  endp

; CmpStr - compare two zero terminated string
;
; Input:
;   SI - pointer to first string
;   DI - pointer to second string
; Output:
;   CF set if *SI < *DI
;   ZF set if *SI == *DI
CmpStr  proc near
    push si
    push di
;
cpst20:  cmpsb ; compare the next byte of
    jnz cpst90 ; the two strings
    cmp byte ptr [si-1],0 ; check for end of string
    jnz cpst20 ; and repeat if not end
;
cpst90:  pop di
    pop si
    ret
CmpStr  endp

_TEXT  ends

end start
DOS System Data Structures

DOS PSP Data Structure

The PSP (program segment prefix) is a 256 (100h) byte data structure at the beginning of the memory image of each running DOS program. DOS creates this structure as part of the process of loading a program into memory for execution. The PSP contains information that DOS uses to control the execution of the program, and to provide information to the system.

The principle elements of the PSP that the programmer is concerned with are:
- the command line string
- and the pointer to the environment block.

The command line string is a count byte followed by up to 126 characters taken from the command line that caused the program to be executed. The DOS command interpreter parses the command name from the command line. The rest of the command line following the command name is copied to this buffer in the last half of the PSP. The command string will be terminated by the carriage return (0Dh) character that terminated the line that the user typed.

The environment pointer is the segment address of the segment that contains this program’s environment.

Environment Table

The environment is a series of zero terminated strings.

- Each string is of the form “VAR=VALUE”.
- Each environment string is terminated by a byte of 0.
- The entire environment buffer is terminated by an additional byte of 0.

The environment strings are used by various programs to provide configuration information to modify their actions. A program can parse the environment table looking for a particular string, and then use the information found to configure itself.

For example: The DOS command parser uses the PATH environment variable to specify a sequence of directories to be searched for programs when processing user commands.

Programs such as compilers and linkers often look for the environment variables INCLUDE and LIB to specify directories to be searched for include files and runtime libraries.
File Table

In an MS-DOS system, there is a **single system file table that keeps track of the information needed to access all open files in the system**.

Each **program that is running has its own process file table**. The process file table **contains indexes into the system file table**.

By default, the process file table is stored in the program’s PSP. However, this **default file table only has room for twenty entries**.

If a program wants to be able to have **more than twenty files open simultaneously**, **it is necessary to relocate the default process file table**. This is done by **allocating a block of memory using DOS function 48h**, copying the **process file table data from the PSP into the allocated memory block**, and then setting the appropriate variables in the PSP to point to the new file table memory.

DOS Memory Management

**DOS Memory Arena**

DOS maintains a free memory pool containing all free memory in the system. Initially, when the system boots all of free memory is contained in a single large block. In this memory pool, **all memory blocks are organized on paragraph boundaries and are sized in paragraphs (16 bytes ~ segment boundaries)**. Essentially, this means that **block addresses are segment addresses and need to be loaded into a segment register to access the memory block**.

DOS provides three explicit memory management functions:

**Fcn 48h - Allocate memory block**

- **On Entry:** BX – number of paragraphs requested
- **On Exit:** AX – segment address of allocated block
- **Errors:** CY set, with error code in AX
  
  BX will contain the size of largest available block

**Fcn 49h - Free memory block**

- **On Entry:** ES – segment address of block to free
- **On Exit:** no return value
- **Errors:** CY set, error code in AX
Fcn 4Ah - Modify memory block

On Entry: AX – new sized requested (in paragraphs)
          ES – segment address of block to modify

On Exit: no return value

Errors: CY set, error code in AX
If error code is 0008h (out of memory), BX will contain the size of the
largest available free block.

When a program loads, DOS allocates and assigns a block of memory of the
size requested in the EXE file header.

By default, the linker sets this to a value that causes all of memory to be
allocated to the program.

In this case, it is necessary to use DOS Fcn 4Ah (Modify memory block) to
shrink the memory allocated to the program to free up some memory before
Fcns 48h (Allocate memory block) can be used to allocate memory.

The program’s PSP address is the address to use for the memory block containing
the program.